July 2021

EVALUATING WORD EMBEDDING MODELS FOR TRACEABILITY

Mahfuza Khatun

Mahfuza Khatun
Louisiana State University and Agricultural and Mechanical College

EVALUATING WORD EMBEDDING MODELS FOR TRACEABILITY

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master's of Computer Science

in

The Department of Computer Science

by
Mahfuza Khatun
B.S., University of Dhaka, Bangladesh, 2014
M.S. University of Dhaka, Bangladesh, 2016
August 2021

Towards peace and growth

ACKNOWLEDGEMENTS

Life is a continuous process. Process can be standstill or dynamic. I am glad to be in a dynamic process. I am fortunate to experience new experiences and learn new things every day. The joy of knowing the unknown, even sometimes knowing properly the very known is the greatest pleasure and the catalyst of my existence. The good thing about joy and happiness is that it is contagious, and it increases exponentially through sharing. From this point on, I can only move forward and put together all the greatness and happiness that I felt, touched, and came across only to share it with others, so that others can also experience what I experienced and grow beyond me to elevate the world a few levels up towards peace, love, and greatness.

My humble gratitude to my advisor, Dr. Carver, who provided her invaluable advice, encouragement, and wisdom with patience. I could not have done this work without her incessant support and guidance. I thank her for letting me in to her warm shed and teaching me to be hamble. I wish to carry along some of it within myself as a gift from her.

I am thankful to Dr. Danissa Victaoria Rodriguez Caraballo. It is my great pleasure to meet her, I admire her positive energy. She helped me through sharing her work, experiences, and knowledge, provided me the guidance and suggestions whenever I sought for it. Thanks to the committee members, Dr. Jinhua Chen and Dr. Wang for their positive feedbacks and valuable suggestions for future improvement of my thesis.

I thank my best friend and better half, Mohammad Imrul Kayes for his endless support and encouragement in my every step forward. I am forever grateful to him for his unconditional love. I thank him for making sure that I have everything for progress; in particular, I am thankful

to him for providing the healthy space that was inevitable for my success. Thank you for being always there for me and listening to my thoughts. Without his continuous support and inspiration this work may not have been possible. I am greatly in debt to world's smartest and cutest puppy, Bruno, who fueled my day-to-day life with his unconditional love! I am immensely grateful to my parents and siblings who have always been the strength and behind the scenes power source of my every little success.

My heart felt gratitude to Frédéric Chopin for one of his masterpieces. It is as if that he knew the song of my soul and tuned the piece exactly for me. Amidst all stressors, it is the perfect magic wand that can sooth my mind within a blink. I thank all those wonderful minds, Carl Sagan, Sarah May, Mark Coleman, Michelle Obama, Yuval Noha Harari for blessing me the perfect companionship that helped me to focus, think, and grow beyond my boundary. Special thanks to Asha Murphy for being part of my support system, for guiding me towards the path of finding peace. The mental strength that I have gained in our regular meetings has changed my life forever. I want you to know that you will always be in my thoughts, I wish you the best!

I acknowledge the Cox Communication Academic Center for Student-Athletes at Louisiana State University (LSU) for their uninterrupted support towards my education from beginning to the end. It is my pleasure to be a part of this organization. It has been a great opportunity to work with the team, each and everyone in the team has touched my life in some way and shed some light on my path forward. Specially I am thankful to the Tutorial center team, with whom I worked in a regular basis and got to learn from each one of them. I cannot

thank enough to my supervisor, Bradley R. Jones for being one of the bests! His constant support and friendly appearance have made it all smooth and easy-going. Thanks to Dr. Louise Bordeaux for willing to be my mentor, for including me in her busy schedule, for sharing her experiences with me. Thanks to all those young minds to whom I have had opportunity to work with. I want to thank them for warming my heart, for giving hope, and letting me be a part of their wonderful journey ahead. Treasuring all these blessings, I truly feel confident to move forward and serve the world.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Traceability link recovery (TLR) is a software engineering activity that helps to ensure software quality and assists with keeping track of changes by establishing links between software artifacts that are a part of the software engineering process, such as requirements, use cases, source code, test cases, and documentation. Software requirement artifacts are typically written in natural language. An Information Retrieval process is frequently used in many software activities, including the TLR activity. Recently, Word Embedding (WE) techniques have been used in many natural language processing tasks as well as in TLR tasks. We investigate the effectiveness of WE techniques in conjunction with the ABC algorithm for automating the TLR process between requirements and source code. The ABC algorithm, which is a metaheuristic search Swarm Intelligence (SI) algorithm that simulates the behavior of honeybee swarms, is useful for solving multidimensional optimization problems. We use a modified ABC algorithm in which the initial population is generated randomly based on the document ID number within the document set boundaries. We use the algorithm to optimize the objective function and find the best links between the requirements and the source code. For our investigation we use three open source pretrained models: Word2Vec, GloVe, and FastText. We experiment with three objective functions that are optimized by the ABC algorithm to find the best possible links between the documents. Our experimentation with three datasets indicates that the three objective functions result in similar success rates. We use precision, recall, and the F1 measure to determine effectiveness for the TLR task. Our results show that the recall is higher than the precision and that the resulting F1 value does not indicate promise for combining word embedding, our three

objective functions, and the modified ABC algorithm as a recommended approach for automating traceability links between requirements and source code.

1. INTRODUCTION

Software engineering (SE) is a continuously evolving field; it evolves because of its built-in nature of incompleteness (Sommerville, 2016). Moreover, it has to fit in with the demands and needs of consumers. According to Lehman's Law, "Software systems have to change if they are to remain useful." (as cited in Sommerville, 10th ed., p. 271). Requirement changes to fit customer expectations require SE to be adaptive. It demands modifications throughout the lifetime of a system. Traceability Link Recovery (TLR) assists software system evolution as modifications are incorporated into system.

More specifically, TLR is defined as a SE task that establishes a link between different software artifacts from high-level (HL) documentation to low-level (LL) source code (Rodriguez & Carver, 2020). It ensures the quality of the product, keep track of changes, and helps to analyze change impact. It is vital for safety critical products and imperative for bug localization and feature location tasks. Despite its importance, it can be a very time and labor consuming task, motivating researchers to invest in studying how to automate the task of evaluating software links.

Artifacts, including requirements, use cases, test cases, and design documentation are generally written using natural language; therefore, Information Retrieval (IR) processes are used in many software engineering tasks including the TLR task (Antoniol, Canfora, Casazza, De Lucia, & Merlo, 2002). The IR probabilistic, Latent Semantic Indexing (LSI) and Vector Space Model (VSM) are among the most popular IR techniques that are used for this task. An IR model is often used as a base model with other machine learning (ML) techniques, such as learning to

rank (LtR) or optimization algorithms, in order to improve the overall success in evaluating links. In such an endeavor, (Rodriguez & Carver, 2020) combine IR with the ABC (Artificial Bee Colony) algorithm to accomplish link recovery. Their investigation provides an encouraging result in Precision, Recall, and F1 measure, which they hypothesize could be improved even more with additional tuning (e.g., experimental set up change, parameter adjustment). They found that the ABC implementation achieves better Recall when compared to other methods.

Recently, word embedding (WE) has earned popularity in many natural language processing (NLP) tasks. It is also used in TLR tasks, and it has been shown to perform significantly better than the traditional IR techniques (Tian, Cao, & Sun, 2019; Zhao, Cao, & Sun, 2018; Wang et al., 2019; Bella et al., 2019). The success of WE over traditional IR techniques lies in its ability to carry the semantic meaning of the words. Unlike existing IR techniques, WE values the order of the words in a context and resolves the lexical gap problems (Zhao et al., 2018).

Observing the success of the ABC algorithm in the TLR task (Rodriguez & Carver, 2020), we were motivated to further investigate using the WE model along with the ABC algorithm. In our experiment, we use open source pretrained WE models Word2Vec, GloVe and FastText models.

We apply the ABC algorithm following the (Rodriguez & Carver, 2020) implementation. Instead of the *TFIDF* (term frequency-inverse term frequency) based weighted cosine similarity function, we use three different similarity functions, which we call sim_1, sim_2, and sim_3. Using the available pretrained WE models, we learn the term vectors. Then, we learn the

document vector representation by taking the average of term vectors contains in that document. In sim_1, we calculate the cosine similarity between these documents. In addition to the similarity measure used in sim_1, we find another similarity score that is the value given by the number of common terms in two documents (HL and LL documents) divided by the total terms in their combined document (Bella, Creff, Gervais, & Bendraou, 2019). The sim_2 is a linear combination of these two scores weighted by an empirical parameter. In sim_3 we find the HL document vectors in the same way as in sim_1, but the LL document vectors are weighted using their *TFIDF* score (Cheng, Yan, & Khan, 2020). Finally, the ABC algorithm is used to optimize these objective functions. We organize our whole investigation around the following research questions:

RQ1: How does performance vary with each objective function?

RQ2. How does performance vary with different word embedding (WE) pretrained models?

RQ3. How do a WE based objective function and an ABC combination perform in TLR task automation?

In Section 2 we review the related literature, Section 3 discusses word embedding, Section 4 describes the dataset and pretrained models, Section 5 explains the methodology of the overall experiment, Section 6 includes the experimental setup along with the detail experimentation process, and Section 7 contains the results and discussions. In Section 8 we conclude our work and provide future research paths.

2. LITERATURE REVIEW

In an endeavor to reduce the lexical gaps between the software artifacts written in natural language (search queries) and source code, (Ye, Shen, Ma, Bunescu, & Liu, 2016) first use word embedding in the software engineering text retrieval task. They learn the word embedding using word2vec Skip-gram model (Mikolov, Chen, Corrado, & Dean, 2013b) on the documents that contain both high-level and low-level languages, such as API documents, tutorials, and bug reports. Then they measure the semantic similarity between word vectors learned from the embedding using the cosine similarity measure. To measure the document similarity, they use a slightly modified version of the Mihalcea et al.'s (Mihalcea, Corley, & Strapparava, 2006) approach, where they first calculate the similarity between each word *w* in a document *S* (bag-of-words) to any word w' in another document *T* (in bag-of-words representation) and use the maximum value. This relation is expressed in the following Equation (1):

$$sim(w, T) = \max_{w' \epsilon T} sim(w, w') \tag{1}$$

where the word-to-word similarity is simply the inner product of their learned vectors shown in Equation (2):

$$sim(w_s, w_t) = cos(w_s, w_t) = \frac{{w_s}^T w_t}{||w_s||||w_t||} \tag{2}$$

To achieve a better result, they first define a set of words with positive similarity:

$$P(T \rightarrow S) = \{w \epsilon T | sim(w, S) \neq 0\}$$

excluding words with no embedding or that simply do not appear in the target document. They name these modified similarities as asymmetric similarities, which are calculated by taking the sum of similarities between words in a document and the entire bag-of-words in another document divided by the number of words in the previously defined set of words with positive similarity. These computations are given in Equations (3) and (4):

$$asymmetric\_sim(T \rightarrow S) = \frac{\sum_{w \epsilon P(T \rightarrow S)} sim(w, S)}{|P(T \rightarrow S)|} \quad (3)$$

$$asymmetric\_sim(S \rightarrow T) = \frac{\sum_{w \epsilon P(S \rightarrow T)} sim(w, T)}{|P(S \rightarrow T)|} \quad (4)$$

Finally, the symmetric similarity is computed by summing the two asymmetric similarities as defined in Equation (5):

$$symmetric\_sim(T, S) = asymmetric\_sim(S \rightarrow T) + asymmetric\_sim(T \rightarrow S) \quad (5)$$

They improve the performance of bug localization tasks, as well as for API code retrieval tasks (Ye et al., 2016).

In another API code retrieval task improvement method proposed by (Nguyen, Nguyen, Phan, Nguyen, & Nguyen, 2017), they combine traditional IR with Word2Vec to achieve better accuracy in code retrieval compared to the simple IR or simple Word2Vec approach. They first find the two similarity scores using the Word2Vec model used in (Ye et al., 2016) Equation (5) and the rVSM (revised Vector Space Model) model proposed in (Zhou, Zhang, & Lo, 2012). The rVSM model presented in Equation (8) differs from the classical VSM model by providing

higher preference to larger documents than smaller ones. To increase the ranking score of larger documents, rVSM model includes a logistic function Equation (9) in the VSM model. In this investigation, the final similarity is computed using Equation (7), which is measured by taking the linear combination of the similarity scores (rVSM score and Word2Vec score) weighted by a parameter (α) that is calculated based on the Jaccard similarity of the two documents (HL and LL documents). Jaccard similarity is calculated using Equation (6):

$$Jaccard(T,S) = \frac{|T \cap S|}{|T| + |S| - |T \cap S|} \tag{6}$$

where $\alpha = \begin{cases} 0, & if\ norm_{jaccard(T,S)} \leq emperical\ value \\ 1, & otherwise \end{cases}$, which means that α is 0 when the Jaccard similarity is less than or equal to an empirically chosen threshold value, otherwise it is 1.

$$sim(T,S) = \alpha * norm\_rVSM(T,S) + (1 - \alpha) * norm\_Word2Vec(T,S) \tag{7}$$

where the rVSM score and Word2Vce score are the normalized value calculated using Equation (8) and Equation (5) respectively.

$$rVSM(T,S) = g(\#term) * cos(T,S) \tag{8}$$

The function g(#terms) in Equation (8) is defined by Equation (9):

$$g(\#term) = \frac{1}{1 + e^{-N(\#terms)}} \tag{9}$$

where *#term* represents the number of terms in a given document. In Equation (9), *N* is a normalization factor calculated by using Equation (10).

$$N = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{10}$$

where x is any data in a dataset, x_{max} and x_{min} are the maximum and minimum data in that dataset, respectively.

In a TLR task, (Tian et al., 2019) adopt pre-trained word embedding. They consider the out of vocabulary (OOV) words; OOV words are those words that are not found in the pretrained embedding model. They argue that OOV words are important factors for TLR tasks, as they are usually the named entities, technical terms or compound words in the source code. They define a set of OOV words, oov_set along with the Word2Vec word set, $w2v\_set$ (words with embedding) where

$$oov\_set = \{word | word \, \epsilon \, V \cap word \notin w2v\_set\}$$

$$w2v\_set = \{words \; that \; exist \; in \; the \; embedding\}$$

where *V* represents the Vocabulary. Then they map each document into the VSM model based on the OOV set, which is build using the term-frequency (*TF*) and inverse document frequency (*IDF*) as defined by Equation (11), (12), and (13), respectively.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{11}$$

$$IDF_i = log \frac{D}{1 + |\{j: i \epsilon d_j\}|} \tag{12}$$

$$TFIDF = TF_{i,j} * IDF_i \quad (13)$$

where $n_{i,j}$ = number of times i^{th} word appears in document d_j, $\sum_k n_{k,j}$ = number of time i^{th} word appears in D, D is the number of total documents in the dataset, and $\left|\{j: i \epsilon d_j\}\right|$ = number of words contain in the document d_j. To calculate the cosine similarity of the documents with the embedding set, they measure the average of the word vectors in each document to get the document vectors given by Equation (14).

$$T_{w2v} = \frac{\sum_{i=1}^{m} w_i}{m} \quad (14)$$

where w_i is a word in $w2v\_set$. Then the final similarity is computed using Equation (15):

$$sim(T, S) = \alpha * cos(S_{w2v}, T_{w2v}) + (1 - \alpha) * cos\ (S_{oov}, T_{oov}) \quad (15)$$

In Equation (15), the cosine similarity of the words in *oov_set* and *w2v_set* are adjusted with a parameter α which ranges between 0 and 1, ($0 < \alpha < 1$). The value of α is empirically chosen to achieve higher precision. Additionally, they use the machine learning technique, LtR (Learning to Rank), for better accuracy. Their results indicate that this integration of word embedding with LtR works better in their TLR task compared to the single word embedding implementation.

In a similar approach to (Tian et al., 2019), (Xinye Wang, Cao, & Sun, 2019) receive a comparable outcome. In this TLR task they both improve their overall result by introducing LtR.

However, (Xinye Wang et al., 2019) follow a varied similarity method. Instead of defining a OOV word set, they define a set for key words, called *IPTs*. *IPTs* are the top n% words in the whole corpus, which is measured using the same *TFIDF* method as in (Tian et al., 2019). For the final similarity measurement, they consider the (Ye et al., 2016) similarity method with a weighted word-to-word similarity measurement. They first measure the cosine similarity between word vector pairs learned from pretrained-embedding and then weight the similarity result with a harmonic parameter, r (>1), and a set of two thresholds, δ_l ($0 < \delta_l < 0.5$) and δ_h ($0.5 < \delta_h < 1$) using the Equation (16):

$$sim_{mod} = cos(w, w') - \left[\frac{(0.5 - cos(w, w'))}{r}\right] \quad (16)$$

where both words w and w' belongs to *IPTs* and $cos(w, w') < \delta_l, \delta_h < cos(w, w')$. The rest of the similarity measurement process is similar to (Ye et al., 2016).

A novel approach called WELR is introduced by (Zhao et al., 2018) in their TLR task. This method is based on the combination of word embedding and LtR methods. Motivated by the success of query expansion (QE) in IR tasks, they implement QE in this experiment along with the *IDF* weighting strategy. They use *IDF* instead of *TFIDF* to focus only on the common terms and to minimize the process time. Initially they form an expansion word set with top (*topn%*) words based on their *IDF* weight from document *S*. Then the set is expanded with similar words (synonyms) as represented in Equation (17):

$$QE\_SET_w = \{word | sim_{w2w}(w, word) > a\} \quad (17)$$

where $\{w \epsilon Expansion\_set\}$ and parameter *a* is the similarity threshold used to remove less important words. Both *topn%* words in the expansion set and parameter *a* are obtained empirically, depending on the dataset. Next, they calculate the asymmetric document similarity with a modified version of (Ye et al., 2016). Similarity between a word *w* and a document *T* is calculated somewhat differently than Equation (3) (Ye et al., 2016). They add an additional term defined in Equation (18) that gives the average term similarity for words belonging to the query expansion set:

$$g(w, w') = \frac{\sum_{w_k \epsilon QE\_SET_w} l_k * sim_{w2w}(w_k, w')}{|QE\_SET_w|} \quad (18)$$

where , $l_k = sim_{w2w}(w, w_k)$. The modified equation is illustrated in Equation (19):

$$sim_{w2w}(w, T) = \max_{w' \epsilon T}\{\alpha * sim_{w2w}(w, w') + (1 - \alpha) * g(w, w')\} \quad (19)$$

In Equation (19) the parameter α is empirically chosen and tuned with a step of 0.01 to achieve an optimal result. If a word is not in the $Expansion\_set$ then Equation (3) is used to calculate the word-to-document similarity. These values are then used to calculate the asymmetric similarity. Finally, another threshold is analytically defined to filter the ranked list with most similar documents followed by the LtR implementation.

With an aim to reduce false positive link generation in their TLR task, (Bella et al., 2019) present a different approach. Their model is called Aggregation Trace Links Support (ATLaS),

which is based on the clustering hypothesis that combines various methods of IR and NLP techniques (such as word and sentence embeddings). They present an empirical evaluation of the model based on an industrial case study. The inputs of this framework are HL requirements documents and LL models in XML Metadata Interchange (XMI) format, which outputs a list of traceability links along with their confidence measures. The inputs are preprocessed using basic NLP techniques to prepare for the syntactic and semantic measure computation. The syntactic measure is computed using LSI (Xiaobo Wang, Lai, & Liu, 2009), LDA (Panichella et al., 2013) and VSM (Niu & Mahmoud, 2012) scores, and for the semantic measure they use three similarity measures. In these similarity scores they use Word2Vec and GloVe pretrained models to learn word vectors and to build a dictionary of synonymous words and phrases. These three similarity scores are based on the Naïve Satisfaction Method (Holbrook, Hayes, Dekhtyar, & Li, 2013) presented in Equations (20), (21), and (22). The first similarity score, *S1* is obtained by direct implementation of the Naïve Satisfaction Method. In the second similarity score, *S2* verbal phrases and nouns are considered to obtain the score, instead of words. Finally, the third score, *S3* is a variation of the *S1*, and obtained after excluding the less impactful words. They assume that some frequently appearing words such as, "shall", "system" are less useful, the same as stop words; hence, they filtered out those non-impactful words based on an empirically defined frequency threshold.

$$S1 = \frac{N_{common\ terms}}{N_{total\ terms}} \tag{20}$$

$$S2 = \frac{N_{common\ noun\ phrases}}{N_{total\ phrases}} \tag{21}$$

$$S3 = \frac{N_{common\ impactful\ terms}}{N_{total\ terms}} \tag{22}$$

Finally, a confidence matrix is build using all these syntactic and semantic scores. A weight is assigned to each of the semantic similarity scores to overcome the lower performance of syntactic measures. Based on the matrix associated with each requirement model pair, the true links are generated.

In bug localization performance improvement research, (Cheng et al., 2020) combine IR techniques, rVSM and WE to find similar documents. They use DNN (deep neural network), a machine learning technique to integrate these two similarity matrices (rVSM and WE). They compare their result with five existing methods BugLocator (Zhou et al., 2012), LtR (Ye, Bunescu, & Liu, 2014), LtR based on Word Embedding (Ye et al., 2016), and two deep neural network learning based methods DNNLoc (Lam, Nguyen, Nguyen, & Nguyen, 2017) and DeepLoc (Xiao, Keung, Mi, & Bennin, 2018). They indicate that their method improved upon the existing ones and achieved better statistical significance. This method calculates surface lexical similarity and semantic similarity between bug report and source code. To calculate the surface similarity, they redefine the well-known VSM into rVSM, which is weighted cosine similarity times the length factor, presented in Equation (8). The length factor is added to reduce the noise in larger source code files, same as in (Nguyen et al., 2017). Semantic similarity is calculated separately for bug report *T* and source code *S*. For bug report files, each document

vector is the mean of all word vectors (vector learned from the embedding) in the document, and the source code document vector is the sum of all words in the document multiplied by their *TFIDF* weight, divided by the length of document given by Equations (23) and (24):

$$S' = \frac{1}{|S|}\sum_{i \epsilon S} w_{i,S} * TFIDF_{i,S} \tag{23}$$

$$T' = \frac{1}{|T|}\sum_{i \epsilon T} w_{i,T} \tag{24}$$

They do the same measurement for each method in the source code document. Next, the similarity results are calculated by taking the maximum from each set computed by Equations (25) and (26):

$$SurfaceSim(T,S) = max\ (\{rVSM(T,S)\} U \{rVSM(T,m) | m \epsilon S\}) \tag{25}$$

$$SemanticSim(T',S') = max\ (\{cos(T',S')\} U \{cos(T',m') | m \epsilon S\}) \tag{26}$$

where *m* and $m'$are the methods in *S*. Finally, they integrate the similarity measures using a DNN (deep neural network) method.

The TLR task is considered as a combinational problem by (Rodriguez & Carver, 2020); hence, they applied the ABC optimization algorithm. The search space for this problem is defined by the set of document pairs (*T*, *S*). Then the ABC algorithm is used to find the best solution from that search space that optimizes an objective function. They define their objective function as the weighted (by *TFIDF*) cosine similarity measure between the documents, presented in Equation (27):

$$SemSim(T,S) = \frac{1}{m}(\sum_{i=1}^{m} cos\ (weight(T,S))) \quad (27)$$

They first preprocess the requirements and source codes separately. The *TFIDF* model is used to find the document vectors. Then, they measure the cosine similarity of each document pair. The final similarity score is a value obtained by taking the sum of all the cosine similarity scores divided by the total number of document pairs in a dataset, as shown in Equation (27). Following this method, they achieve a high Precision and Recall values.

We summarize our literature review in Table *1*. This Table *1* presents the various research methods involved in the literature to perform code retrieval, bug localization and TLR tasks. We include methods applied to our current research on TLR. Motivated by the literature review, we use WE based models in conjunction with an ABC model in our investigation to perform a TLR task. Instead of the weighted TFIDF based model as used in (Rodriguez & Carver, 2020) we use WE based models to investigate the effectiveness for the TLR task.

Table 1. Summary of Literature Review

Literature	Task	Method
Ye et al. (2016)	Bug localization and API Code retrieval	WE based model
Nguyan et al. (2017)	API Code retrieval	Combined rVSM model and WE based model
Cheng et al. (2020)	Bug localization	Integrated rVSM and WE based DNN model
Tian et al. (2019)	TLR	Combined WE and TFIDF based model
Wang et al. (2019)	TLR	WE based model weighted by a harmonic parameter
Zhao et al. (2018)	TLR	WELR: Weighted (by IDF) WE based model and LtR model
Bella et al. (2019)	TLR	ATLaS: Clustering hypothesis-based model
Rodriguez & Carver (2020)	TLR	Weighted TFIDF based model and ABC model
Khatun & Carver (2021)	TLR	WE based models and ABC model

3. WORD EMBEDDING

The simplest way to achieve word to vector representation is the one-hot representation. The vector of a given word type (a distinct word) is encoded by setting that element as 1 and the rest of the elements in the vocabulary as 0. The dimension of these vectors is equal to the corresponding vocabulary size. With this increased dimension, the word to vector transformation becomes very challenging and costly. Another drawback of this representation is that it does not carry any semantic meaning of the words. Each word is embedded in isolation and contains the same number of 0's and a single 1; hence, the resultant vectors neither provide any information about each other nor about themselves.

N-gram is another popular language model based on the Markov Model (Peter F. Brown, DeSouza, Mercer, Pietra, & Lai, 1992). It takes a probabilistic approach in language modeling, where words are the atomic units of the model (Mikolov et al., 2013b). The concept is to consider a sequence of *N-1* words to predict the next possible word in that sequence. Therefore, the N-gram model is built by counting the occurrences of a sequence of *N* words in a given corpus and assessing the related probabilities of the words. If a model counts the number of occurrences of a single word without looking into any previous words, then it is called a unigram model. Likewise, a bigram model predicts a word based on the word right before it; whereas a trigram model looks into the previous two words then predict the third possible word in that sequence. Again, this process does not carry any information about the words or their order; because words are represented as the indices in a vocabulary (Mikolov et al., 2013b). Moreover, N-gram is known as a sparse model since the word prediction depends largely on the training

dataset. If a certain word is not in the training set, it gets a zero-probability score. Consequently, the training requires a large number of data to build a quality model, which increases computational overhead.

Word embedding (WE) handles the 'curse of dimensionality' (refers to all the problems that come with oversized dimension) by utilizing a neural network-based language model (NNLM) (Bengio, Ducharme, & Vincent, 2001). WE also outperforms the N-gram model in a larger context (greater than trigram) (Bengio et al., 2001). It replaces the discrete vector representation (as in one-hot) with distributional vector representation of words. It is developed based on the hypothesis that words with similar context have similar meaning (Zheng, Shi, Guo, Li, & Zhu, 2017). Each word vector is learned by considering that each element in the vector space (represented by the vocabulary) takes part in forming that vector.

3.1. Word2Vec

The Word2Vec WE method is presented by Mikolov et al. (Mikolov, Chen, Corrado, & Dean, 2013a). They indicate that the model can learn a word vector with better dimensionality at significantly lower computational costs than N-gram. This cost optimization is achieved by removing the non-linear hidden layers, leaving only the projection layer in the neural network. The architecture is based on two separate steps, consisting of a continuous word vector learning step (here they use one-hot method) and a training step (objective is to maximize the conditional probability) that trains the N-gram NNLM on that learned vector. They propose two models, Continuous Bag-of-Words (CBOW) and continuous Skip-gram (SG) models. These models are elaborately explained in (Goldberg & Levy, 2014) and (Rong, 2014). The concept is that the

CBOW predicts the current word vector based on the input context vectors (like bigram model), whereas SG predict the context vectors of an input word vector. Figure *1* represents these two architectures.

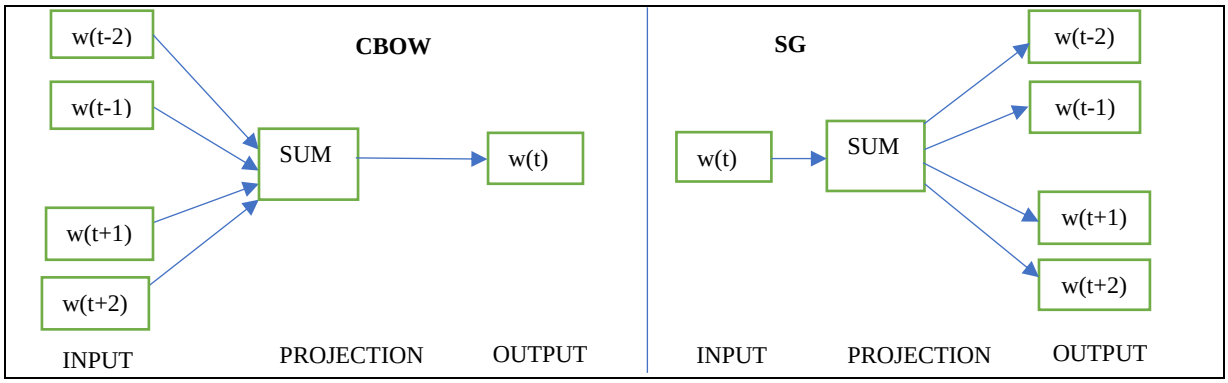

Figure 1. CBOW and SG model architectures

In their extended work with the SG model (Mikolov et al., 2013b), they replace the hierarchical SoftMax with Negative Sampling method, and introduce the subsampling of frequent words. They find that these changes improve the obtained vector (both word and phrase vectors) quality and speed up the overall process. They find that compared to other concurrent neural network-based models, Skip-gram outperforms in word analogy tasks. Additionally, they find that the time complexity remains significantly low even with their implementation of larger (two to three times) training datasets. With these improvements, the Skip-gram model achieved state-of-the-art recognition in word embedding. Later, they publish this trained model as Word2Vec model for public use. Since then, researchers have implemented the Word2Vec model in many SE tasks,

including the TLR task (Guo, Cheng, & Cleland-Huang, 2017; Nguyen et al., 2017; Tian et al., 2019; Xinye Wang et al., 2019; Zhao et al., 2018).

3.2. GloVe

Global Vector (GloVe) for word representation captures a global corpus statistic, based on the word-to-word cooccurrence (Pennington, Socher, & Manning, 2014). Pennington et al. (Pennington et al., 2014) claim that this model performs better in analogy tasks, by combining two popular methods, the global matrix factorization method and Word2Vec model. It starts with building the word-word cooccurrence matrices, which tabulates the number of times a word w_j occurs in a given context of word w_i, and the sum of the number of times any other word appears in that same context of word w_i. Next the probability of a word w_j appearing in the context of a word w_i is measured by taking their ratio. Then the ratio of the probabilities is used instead of the probabilities as the base for learning word vectors. They argue that the probability ratio performs better not only in separating the relevant words from the irrelevant ones, but also in separating two relevant words. The cooccurrence matrix is then factorized following the LSA model (Scott, T, W, K, & Richard, 1990) and used as a baseline model. They assert that this model does a better job for the following tasks: word similarity, named entity recognition, and word analogy tasks. They also compare this model with the Word2Vec model and assert that GloVe consistently outperforms Word2Vec.

3.3. FastText

According to (Bojanowski, Grave, Joulin, & Mikolov, 2017), many popular continuous word representations lack morphological meaning by assigning distinct vectors to each word.

They propose the FastText model incorporating the N-gram in the SG model (Mikolov et al., 2013b). They represent each word as a bag of character N-grams (instead of one-hot representation). Hence, each word vector becomes the sum of these character N-gram vectors comprising that word. They start by building a set of character N-gram. They assign a special boundary symbol "< >" representing the character N-grams of a word. This boundary symbol separates the suffixes and prefixes of a word from other character sequences. They also consider each word as a special sequence and wrap them with the same boundary symbol. The final set of N-grams includes all these character N-grams as well as the special sequences. This N-gram representation includes $3 \leq N \leq 6$. Using the Fowler-Noll-Vo (FNV-1a variant) hashing function, they minimize the memory requirement of the model, where they map N-grams to integers, limiting it from1 to $2x10^6$. Therefore, words are described by their specific indices in the dictionary along with the set of hashed N-grams they consist of. They state that this model is capable of learning reliable representations for rare words through its shared representation across words (an out of vocabulary word can be represented by summing up the character N-grams consisting of that word). Consequently, they assert that the consideration of the subword information has significantly improved their model, and it achieved state-of-the-art performance in word analogy and similarity tasks. Furthermore, they indicate the model as a simple and fast model, requiring no additional preprocessing step.

4. ABC ALGORITHM

The ABC algorithm was developed by observing the social behavior of honeybee swarms (Karaboga & Basturk, 2007). The artificial bee colony model simulates the behavior of the real bee swarms, and it is used for solving multidimensional optimization problems. In this model there are three groups of bees: employed bees, onlooker bees, and scout bees. Half of the total population in a colony are employed bees and the rest are onlookers. The number of food sources around the hive is equal to the number of assigned employed bees, which means that each employed bee is assigned to exactly one food source. Once a food source is abandoned by the bees, the associated employed bee becomes a scout bee. The scout bee then randomly searches for new potential food sources and memorizes their locations to share with other bees in the hive. The information sharing takes place in a dance area in the hive via waggle dancing of the bees. The onlooker bee on the dance floor carefully observes the dances and employs herself to a profitable food source.

The position of a food source represents a possible solution of the optimization problem, and the quality of the solution is determined by the nectar amount of the source. Therefore, the number of solutions in a population is equal to the number of employed bees or onlooker bees. The algorithm first generates a randomly distributed initial population based on the food source positions. Then the employed, onlooker, and scout bees initiate their repetitive search cycle. An employed bee memorizes the position of a new food source only if it provides a better nectar amount (fitness) than the existing source in her memory; otherwise, she keeps the information of the old source. Once all the employed bees in a hive complete their search process, they gather in

the waggle dance area and share their knowledge about the source position and the corresponding nectar value through waggle dancing. An onlooker bee observes their dance and finds a food source based on a probability value associated to its nectar amount. The employed bee modifies the source position in her memory and checks its nectar value. If the nectar value of the new one is higher than the previous one, then the bee forgets the old one and memorizes the new one. A more detail explanation of the steps involved in the ABC algorithm is provided below.

4.1. PARAMETER INITIALIZATION

The ABC algorithm has four control parameters that need to be initialized in the beginning: the solution number (*SN*), the maximum cycle number (*MCN*), and the value of *limit.* The number of solutions (*SN*) is equal to the number of employed bees or onlooker bees in a hive. The parameter *MCN* controls the maximum number of food generation. The parameter *limit*, called the limit for abandonment (Karaboga & Basturk, 2007), is the maximum number of attempts that an employed bee gets to find an improved food source before abandoning the current food source.

4.2. INITIAL POPULATION GENERATION

An initial population of size *SN* is generated following Equation (28). Each solution vector $X_{i,j}$ is D-dimensional; dimension *D* is equal to the number of optimization parameters:

$$X_{i,j} = LB_j + rand(0,1) * (UB_j - LB_j) \quad (28)$$

where $i \in \{1, 2, \ldots, \mathrm{SN}\}$, $j \in \{1, 2, \ldots, \mathrm{D}\}$, rand (0, 1) generates random numbers between 0 and 1, and LB_j, UB_j are lower and upper values of the decision variable $X_{i,j}$.

4.3. EMPLOYED BEE PHASE

Each employed bee visits its assigned food source and memorizes its position and nectar value. After the waggle dance phase, she modifies the information in her memory using Equation (29) to produce a new potential food source. She applies a greedy selection method to find the best possible source at the time. If the old source has a higher nectar value than the new one, then she keeps the old source information in the memory without any change; otherwise, the new source has a high nectar value. She then forgets the old food source and memorizes the new source information.

$$V_{i,j} = X_{i,j} + rand(-1, 1) * (X_{i,j} - X_{k,j}) \tag{29}$$

where $k \in \{1, 2, \ldots, \mathrm{SN}\}$ is randomly determined and has to be different than i, and rand (-1, 1) is a random number between -1 to 1. This candidate food source generation is controlled by the parameter *MCN* so that the food sources are limited to the neighboring area of the hive.

4.4. ONLOOKER BEE PHASE

An onlooker bee carefully observes the waggle dancing of employed bees. Based on the nectar value associated to each source, the bee calculates the probability of the source applying Roulette Wheel Selection method presented in Equation (30):

$$P_i = \frac{fit_i}{\sum_{j=1}^{SN} fit_j} \tag{30}$$

where, P_i is the probability of food source X_i, fit_i is the fitness cost (nectar value) of X_i at ith position, and $\sum_{j=1}^{SN} fit_j$ is the total fitness cost of all the solution pairs at this position.

4.5. SCOUT BEE PHASE

When the food source is not improved within the defined *limit* (cycle number), the onlooker bee abandons it. The associated employed bee becomes a scout bee. This scout bee roams around in search for new resources and follows the same procedure as in the initial population generation step.

4.6. TERMINATION

The above steps are reiterated until termination condition is met. This condition is expressed with the parameter *MCN*, empirically allocated in the initialized stage for convergence.

5. DATASETS AND PRETRAINED MODELS

5.1. DATASETS

For our experiment we choose three datasets, EasyClinic, eTour, and EBT. These datasets are recurrently used in literature for many software engineering tasks, especially in the TLR task of bug localization. We collect the datasets from the Center of Excellence for Software and Systems Traceability (CoEST) website (coest.org), an open-source resource for traceability research. All of our datasets include the requirement/use-cases and source code (represented by classes) documents along with their trace links, which are needed to validate our results. These datasets are presented in Table *2*.

The eTour project is a tour guide project, EasyClinic is a project for hospital management, and Event Based Traceability (EBT), a traceability software built upon event-notification. These systems are developed in Java programming language. The eTour includes 58 use cases, 116 Java source code classes, and 308 trace links between use case to class. EBT has 40 requirements, 50 source code classes, and 98 true links between them. EasyClinic contains 30 use cases, 20 interaction diagrams, 63 test cases, and 47 class description. In total EasyClinic provides 1388 trace links, which includes 93 trace links between the use case to class description.

Table 2. Dataset Description

Dataset	Description	True links
eTour	• Tour guide project • 58 use cases • 116 code classes • Lines of code 25,011	• 308 trace links from use cases to code classes
EBT	• Event based traceability software • 40 requirements • 50 java source code classes • Contains 2,773 lines of code • 25 test cases	• 98 trace links from requirements to classes • 51 trace links from requirements to test cases
EasyClinic	• Hospital management project • 30 use cases (UC) • 47 code classes (CC) • 63 test cases (TC) • 20 interaction diagrams (ID)	• 93 trace links from use cases to code classes • 63 UC-TC links, 26 UC-ID links, 69 ID-CC links, 83 ID-TC links, 204 TC-CC links, 59 ID-ID links, 144 UC-UC links, 578 TC-TC links, 69 CC-CC trace links

5.2. PRETRAINED MODELS

In our investigation we consider the following three publicly available pre-rained word embedding models: Google's Word2Vec (Mikolov et al., 2013a), GloVe by Stanford (Pennington et al., 2014), and Facebook produced FastText (Bojanowski et al., 2017).

The pretrained Word2Vec model is trained on about 100 billion words and phrases collected from Google News. It includes 3,000,000 word vectors with dimension 300. The file size is 1662 MB. Table 3 depicts a brief description of the pretrained models.

Two different GloVe models are available, one based on Twitter and the other trained on datasets collected from Wikipedia 2014 and Gigaword 5 (6B tokens uncased). They each have a vocabulary of size 400,000 and come with varied file size and vector dimension. For our experiment, we use the Wiki-Gigaword combination-based model, which comes in a file of size 376 MB, and the vector dimension is 300. We convert these models into Genism w2v format before use.

FastText is an extension of Word2Vec created by Facebook's AI Research (FAIR) lab in 2015 (Liu, Chan, Feng, Fulton, & Wu, 2019). FastText includes 999,999 word vectors of dimension 300. The model is trained on dataset Wikipedia 2017, UMBC web base corpus and statmt.org news dataset (16B tokens). The file size is 958MB.

Table 3. Pretrained Models

Pretrained Embedding	Description
Word2Vec	• Trained on about 100 billion words and phrases collected from Google news • Vocabulary size 3,000,000 • Vector dimension 300 • File size 1662 MB
GloVe	• Trained on dataset collected from Wikipedia 2014 and Gigaword 5 (6B tokens uncased) • Vocabulary size 400,000 • Vector dimension 300 • File size 376 MB
FastText	• Trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens) • Vocabulary size 999,999 • Vector dimension 300 • File size 958 MB

6. METHODOLOGY

We provide an overview of the techniques we use from the data preprocessing to the optimization process. Our method involves four separate steps:

1. Preprocess data.

2. Learn word vectors using pretrained WE models.

3. Choose similarity measures.

4. Run the ABC algorithm using the similarity measures.

6.1. DATA COLLECTION AND PREPROCESSING

After data collection, we separate the high-level (HL) textual documents (such as requirements and use cases) and low-level (LL) documents (source code files) for our experimentation. We preprocess the HL and LL documents separately. First, we remove the numeric and non-alpha numeric from the documents, leaving only the meaningful words. We convert those meaningful words into lowercase, remove stop words and tokenize them. In addition to these steps, the LL document's preprocessing requires some more steps because of its varied structure. We strip the multiple white spaces and remove the programming language keywords that are not relevant for our purpose. These preprocessing steps are performed using the highly efficient Genism preprocessing tool.

6.2. VECTOR REPRESENTATION

Once the preprocessing is done, we represent our documents in vector form (feature extraction) using the pretrained word embedding model. First, we load the pretrained model into our workspace, which takes about 2 minutes in Jupyter notebook. Once the pretrained model is

loaded, it is ready to use for learning word embedding from the preprocessed documents. We use three different open source pretrained models (Word2Vec, GloVe, FastText) in our experiment. The goal is to compare the results obtained using each pretrained model.

6.3. SIMILARITY MEASURES

We use three different similarity measures in our investigation. The first similarity measure, **sim_1** is obtained using the cosine similarity between two document vectors. To calculate the document vector (T_{w2v} and S_{w2v}), we first measure the word vectors in a document, then take their average as defined in Equation (34). The second similarity measure, **sim_2**, is a linear combination of the first similarity measure, sim_1 with another similarity score, *S_score* obtained using Equation (35). This **sim_2** is tuned by a parameter α, which varies with the dataset, and is adjusted empirically to achieve the best results. The third similarity score, **sim_3** is also a cosine similarity between two documents; however, in this case, the LL document vectors are learned differently than the HL ones. The HL document vectors (T_{w2v}) are learned using Equation (34). The LL document vectors (S_{w2v_tfidf}) are learned using Equation (36), where we include the *TFIDF* weight of the LL document. These three similarity functions are presented in Equations (31), (32), and (33), respectively.

$$sim\_1(T,S) = \cos(T_{w2v}, S_{w2v}) \tag{31}$$

$$sim\_2(T,S) = \alpha * sim\_1(T,S) + (1-\alpha) * S\_score \tag{32}$$

$$sim\_3(T,S) = \cos\left(T_{w2v}, S_{w2v\_tfidf}\right) \tag{33}$$

where

$$T_{w2v} = \frac{1}{|T|}\sum_{i \epsilon T} w_i \tag{34}$$

$$S\_score(T,S) = \frac{N_{common\ terms}}{N_{total\ terms}} \tag{35}$$

$$S_{w2v\_tfidf} = \frac{1}{|S|}\sum_{i\epsilon S}(w_i * TFIDF_i) \tag{36}$$

In these similarity functions *T* and *S* represent HL and LL documents, respectively. The document vector, T_{w2v} is calculated by averaging the word vectors (*w*) in document *T*. The S_{w2v} is calculated in the same way as T_{w2v}. The value S_{w2v_tfidf} represents the modified vector representation of S_{w2v} in which each word vector is learned using one of the pretrained models and is weighted by the *TFIDF* score of that word. The *TFIDF* score is calculated using Equation (13).

6.4. ABC IMPLEMENTATION

We use the modified ABC algorithm for the TLR purpose as used in (Rodriguez & Carver, 2020). Their objective function is based on weighted cosine similarity, Equation (27). We consider three different similarity functions, sim_1, sim_2 and sim_3 defined correspondingly in Equation (31), (32), and (33) as our objective functions to evaluate the food sources (solution vectors).

The first step of the ABC implementation involves the generation and initialization of the initial population and parameters. We assign unique integer ID numbers to our input data, requirements (HL) and source code (LL). A food source is produced by a list of these integer ID pairs which belongs to HL and LL documents, respectively. These pairs are generated randomly and are controlled by a parameter (*Max_size_sol*), representing the maximum number of pairs in

a food source. The parameter *Max_size_sol* is empirically decided based on the dataset. Since different datasets have different numbers of HL and LL documents, they will have distinctive food source sizes. Table *4* is an example of the food source representation, where HL12 is a requirement document number with an assigned ID number 12, and LL3 is a source code document number with ID number 3.

Table 4. Food Source Representation

HL12	LL3
HL15	LL31
HL39	LL10

Therefore, the initial population of the modified ABC algorithm is an integer vector consisting of the list of random pairs (food source/vector solution). A boundary is set up to keep the random generation inside the domain (the maximum ID numbers of the HL and LL documents). The lower boundary is set to zero (minimum possible ID value), while the upper boundary depends on the maximum ID number of the HL and LL documents in the dataset.

An individual food source has a distinctive value (nectar value) associated with it, which is obtained using the objective function. The solution depends on the quality of this nectar value of the food source. An employed bee is assigned to a food source to collect the nectar from it. The parameter *SN* is defined to properly regulate the number of food sources and employed bees generated at each iteration.

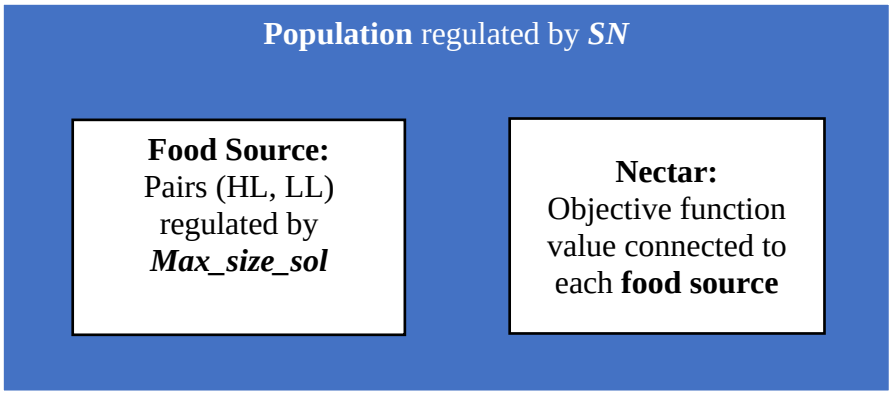

Figure 2. TLR to ABC mapping

The goal is to find a food source that maximizes the nectar value at each iteration. This situation is depicted in Figure *2*.

Each employed bee visits its assigned food source to collect nectar. The mutation operator plays a very important role here. It uses a random mutation factor to decide mutation for a specific pair. Once the employed bee is assigned for a food source, this operator selects three different pairs from the source and applies Equation (37) for mutation (Rodriguez & Carver, 2020):

$$MutatedPair = [X_i] + (randominteger) - [X_n] \quad (37)$$

where, X_i is the current food source, and X_n is a neighboring food source randomly selected. This mutation is regulated by another function to check the boundary conditions of the ID pairs. Crossover and mutation act together to produce new possible solutions for an employed bee.

With the information gathered from the employed bees, the onlooker bee performs a selection process. For this purpose, we use Equation (30), as used in the original ABC algorithm. Using this probability measure, the onlooker bee chooses the quality food source at a given position. Then, it uses the mutation operator Equation (37) to produce a new set of food sources.

The employed bee becomes a scout bee when the food source is not improved within a specified cycle number, defined by the parameter *limit*. This parameter is initialized in the parameter initialization step. Then the scout bee generates new random food sources using Equation (28). Eventually, the program terminates when it hits the termination condition, *MCN*, which is also predefined to achieve convergence.

7. EXPERIMENT

The experiment is developed in Python-3.6.9 and runs on a Jupyter notebook. The first step of the experimentation is the preprocessing of the datasets. We use Gensim 3.8.3 for preprocessing and loading the pretrained embeddings. A detailed description of the preprocessing was provided in Section 5.

We evaluate our results based on the IR metrics precision (P) and recall (R). The P value is calculated using Equation (38), which measures the correctness of the result based on the number of true positives (TP) and false positives (FP) obtained. Equation (39) is used to measure the R value, where FN stands for the number of false positive links found. R represents the completeness level of the result.

$$P = \frac{TP}{TP + FP} \epsilon [0,1] \tag{38}$$

$$R = \frac{TP}{TP + FN} \epsilon [0,1] \tag{39}$$

ABC parameter set up: Based on the dataset we set part of the ABC algorithm's parameters initially, which remain unchanged throughout the experimentation. More specifically the upper and lower boundaries for a specific dataset will remain the same throughout the experimentation process. The lower boundary is set to 0 for all of the datasets, and the upper boundaries vary with the dataset. For the EBT dataset we set the upper boundary equal to 40, representing the number of HL documents in the dataset. Similarly, for EasyClinic the upper boundary is set to 29, and for eTour it is set to 57.

7.1. Objective Function sim_1

We use sim_1 as the objective function for the ABC algorithm. The function sim_1 was described in subsection 6.3. For convenience, Equation (31) is presented again here:

$$sim\_1(T, S) = \cos(T_{w2v}, S_{w2v}) \quad (31)$$

We run the ABC algorithm at different *SN* and *MCN* assignments with objective function sim_1. The results using the three different datasets follow.

EBT dataset with sim_1: The results for the EBT dataset with sim_1 are listed in Table *5*. The values shown in bold represent the highest values from each embedding model. We use all three of the pretrained models on EBT. After some experimentation we set the parameter *Max_size_sol* equals to 250, which provides the best result. The highest P score of 0.1875 is obtained using the Word2Vec model, and it is found at 90 *MCN* and 90 *SN* assignments. The corresponding R score is 0.1837. Using GloVe model, we find the highest P of 0.1739 and the highest R of 0.2449 are found at 100 *MCN* and 50 *SN* number. The highest R score for FastText model was 0.143 with a P score of 0.102 at 50 *SN* and 100 *MCN*.

EasyClinic dataset with sim_1: We list the experimentation on the EasyClinic dataset using sim_1 objective function in Table *6*. The highest values obtained from each WE model is shown in bold. We use all three of the pretrained model for this dataset as well. First, we find the parameter *Max_size_sol* that works best for the EasyClinic dataset. We find that 150 provides the best outputs. With the Word2Vec model, assigning the *Max_size_sol* to 150 and parameter *SN* and *MCN* to 50, the highest P score obtained is 0.1023 with the corresponding R of 0.0968. The

result found with GloVe model has the highest P score of 0.0877 with R score of 0.106, which is obtained at 50 *SN* and 100 *MCN* assignments.

Table 5. EBT with sim_1 result

Embedding	Max_size_sol	SN	MCN	P	R
Word2vec	250	50	50	0.1243	0.2245
		60	60	0.1133	0.1735
		70	70	0.1277	0.1837
		80	80	0.1129	0.1429
		90	**90**	**0.1875**	**0.1837**
		100	100	0.1667	0.1837
		110	110	0.1494	0.1327
		150	150	0.1714	0.1224
GloVe	250	50	50	0.1193	0.2143
		90	90	0.152	0.1939
		100	100	0.169	0.2449
		50	**100**	**0.1739**	**0.2449**
		110	110	0.1354	0.1327
FastText	250	50	50	0.073	0.1327
		50	**100**	**0.1022**	**0.1429**
		100	100	0.1034	0.0918

The FastText pretrained model produces lower results than the other two models. The highest P score is 0.034 and R score is 0.043.

Table 6. EasyClinic with sim_1 result

Embedding	Max_size_sol	SN	MCN	P	R
Word2Vec	150	40	40	0.0566	0.0645
		50	**50**	**0.1023**	**0.0968**
		60	60	0.086	0.108
		90	90	0.085	0.086
		100	100	0.063	0.065
		50	100	0.0725	0.0538
GloVe	150	50	50	0.071	0.149
		50	100	0.065	0.054
		60	60	0.058	0.054
		100	**100**	**0.0877**	**0.106**
FastText	150	50	50	0.0213	0.0426
		50	100	0.042	0.032
		100	**100**	**0.0339**	**0.0426**

eTour dataset with sim_1: The results obtained using sim_1 on the eTour dataset are listed in Table *7*, where the highest values are shown in bold. The best result is obtained using the GloVe pretrained model at 400 *Max_size_sol* and 50 *SN* and 50 *MCN*, which scored the

highest P of 0.1138 and R of 0.1234. Both Word2Vec and FastText models have comparable results. Word2Vec produces the highest P score of 0.0962 and 0.0909 R score. FastText has the highest P score of 0.0909 and R score of 0.0974.

Table 7. eTour with sim_1 result

Embedding	Max_size_sol	SN	MCN	P	R
Word2Vec	400	50	50	0.0696	0.0779
		50	**100**	**0.0962**	**0.0909**
		60	60	0.0613	0.0617
		90	90	0.053	0.042
		100	100	0.0882	0.0682
GloVe	400	**50**	**50**	**0.1138**	**0.1234**
		60	60	0.066	0.068
		50	100	0.064	0.058
		90	90	0.088	0.068
FastText	400	**50**	**50**	**0.0909**	**0.0974**
		60	60	0.0501	0.052
		50	100	0.031	0.029
		90	90	0.067	0.055

7.2. Objective Function sim_2

We use the sim_2 as the objective function for the ABC algorithm. A detail description of sim_2 is presented in subsection 6.3, and Equation (32) is repeated here for convenience:

$$sim\_2(T,S) = \alpha * sim\_1 + (1-\alpha) * S\_score \quad (32)$$

Again, we investigate this objective function based on the three datasets. The experimentation is described below, and the results of this investigation are presented in Table *8* for EBT dataset, Table *9* for EasyClinic and Table *10* for the eTour dataset, accordingly. The highest values obtained from each WE model are shown in bold.

Table 8. EBT with sim_2 result

Embedding	Max_size_sol	SN	MCN	ALPHA	P	R
Word2Vec	250	50	50	0.5	0.058	0.102
		50	100	0.5	0.045	0.061
		100	**100**	**0.5**	**0.104**	**0.112**
GloVe	250	50	50	0.5	0.056	0.102
		50	**100**	**0.5**	**0.1447**	**0.1122**
		100	100	0.5	0.1346	0.143
FastText	250	50	50	0.5	0.08	0.123
		50	100	0.5	0.053	0.082
		100	**100**	**0.5**	**0.105**	**0.122**

EBT dataset with sim_2: The highest P value is 0.1447 and R value is 0.112 using the GloVe model by setting the ABC parameter *Max_size_sol* to 250, *MCN* to 100, and *SN* to 50. The parameter α of sim_2 is assigned to 0.5, chosen empirically. We kept α the same for all three of the pretrained models. Both Word2Vec and FastText have similar P and R scores. The best results obtained with the Word2Vec model is at 100 *SN* and *MCN* assignment; the highest R score is 0.112 and R is 0.104. Similarly, the FastText model achieves the highest P score of 0.105 with an R score of 0.122. These results are listed in Table *8*.

Table 9. EasyClinic with sim_2 result

Embedding	Max_size_sol	SN	MCN	ALPHA	P	R
Wor2Vec	**150**	**50**	**50**	**0.5**	**0.0947**	**0.0968**
			100	0.5	0.065	0.0645
		100	100	0.5	0.0714	0.0851
GloVe	**150**	**50**	**50**	**0.5**	**0.0899**	**0.086**
			100	0.5	0.0753	0.052
		100	100	0.5	0.0735	0.0538
FastText	**150**	**50**	**50**	**0.5**	**0.06**	**0.0645**
			100	0.5	0.056	0.055
		100	100	0.5	0.06	0.044

EasyClinic dataset with sim_2: The highest P and R scores are obtained at the same parameter setup for all three of the models; the ABC parameter *Max_size_sol* is set to 150, *SN* and *MCN* both are set to 50, and the sim_2 parameter is set to 0.5, which provides the best result. The Word2Vec model provides the best scores out of all three, with a high P score of 0.0947 and an R equal to 0.0968. GloVe gives the highest P score of 0.0899 and R score of 0.086. FastText provides a P score of 0.06 and R score of 0.0645.

Table 10. eTour with sim_2 result

Embedding	Max_size_sol	SN	MCN	ALPHA	P	R
Wor2Vec	400	50	50	0.5	0.061	0.065
		60	60	0.5	0.052	0.052
		50	100	0.5	0.073	0.071
		100	**100**	**0.5**	**0.087**	**0.071**
GloVe	400	50	50	0.5	0.067	0.075
		60	**60**	**0.5**	**0.073**	**0.075**
		50	100	0.5	0.073	0.071
		100	100	0.5	0.062	0.042
FastText	400	**50**	**50**	**0.5**	**0.084**	**0.064**
		60	60	0.5	0.06	0.062
		90	90	0.5	0.0494	0.042
		100	100	0.5	0.046	0.036

eTour dataset with sim_2: The results obtained using objective function sim_2 with the eTour dataset and the three models are listed in Table *10*. We find that assigning 400 to *Max_size_sol* provides the best result. The GloVe model produces the best results, which is 0.073 as the P value and 0.075 as the corresponding R value. The values are found with 60 *SN* and 60 *MCN* assignments. At 100 *SN*, and *MCN* with sim_2 parameter set to 0.5, a high P score of 0.087 with a R score 0.071 is obtained using the Word2Vec model. A similar result is obtained using the FastText model at 50 *SN* and *MCN*, which provides the highest P of 0.084 with a R of 0.064.

7.3. Objective Function sim_3

We use the three pretrained models on our datasets with sim_3 as an objective function for ABC algorithm. This function is presented in subsection 6.3, Equation (33), and is repeated here for convenience:

$$sim\_3(T,S) = \cos\left(T_{w2v}, S_{w2v\_tfidf}\right) \tag{33}$$

The investigation on the three datasets is presented in Table *11*, Table *12*, and Table *13*; where Table *11* represent the EBT dataset, Table *12* is the EasyClinic dataset, and Table *13* is the eTour dataset. The bold values in the tables represents the highest value found from each WE model.

Table 11. EBT with sim_3 result

Embedding	Max_size_sol	SN	MCN	P	R
Word2Vec	250	50	50	0.09	0.153
		50	100	0.112	0.153
		100	**100**	**0.14**	**0.143**
Glove	250	50	50	0.1356	0.2449
		50	100	0.1544	0.2347
		100	**100**	**0.1869**	**0.2041**
FastText	250	50	50	0.061	0.112
		50	100	0.066	0.092
		100	**100**	**0.104**	**0.102**

EBT dataset with sim_3: The experiment with the EBT dataset shows that the best R score is obtained using the GloVe pretrained model where both *MCN* and *SN* are set to 100 at which the highest P score and R scores of 0.1869 and 0.2041 are obtained, respectively. The Word2Vec and FastText models provide a comparable result. The Word2Vec obtains the highest P score of 0.14 and a R score of 0.143 at 100 *SN* and *MCN* set up. The FastText model provides the highest P score of 0.104 with a R score of 0.102 at 100 *SN* and *MCN*.

Table 12. EasyClinic with sim_3 result

Embedding	Max_size_sol	SN	MCN	P	R
Word2Vec	150	**50**	**50**	**0.101**	**0.0968**
		50	100	0.078	0.0538
		70	70	0.057	0.043
GloVe	150	**50**	**50**	**0.0761**	**0.075**
		50	100	0.056	0.043
		70	70	0.044	0.032
FastText	150	**50**	**50**	**0.036**	**0.032**
		50	100	0.015	0.011
		70	70	0.0299	0.0215
		100	100	0.021	0.011

EasyClinic dataset with sim_3: The EasyClinic dataset obtained the best score with Word2Vec, which is a high P equals 0.1011 and R equals to 0.0968. Using the GloVe model, a high P of 0.0761 is scored with an R of 0.075. For the FastText model, high P and R scores are 0.036 and 0.032, respectively. These scores are found at 50 *SN* and *MCN*.

eTour dataset with sim_3: We find a high R score of 0.081 and corresponding P score of 0.077. These values are obtained with the GloVe model with the parameter setup of 60 for both *SN* and *MCN*, and 400 for *Max_size_sol.* With the Word2Vec model a high P value is found to be 0.0785 and the corresponding R score is 0.0747, which are obtained at 50 *SN* and 100 *MCN*

set up. With the same *SN* and *MCN* assignment, the FastText model obtained a comparable result as obtained in Word2Vec. For the FastText the highest R score is 0.071 and the corresponding P score is 0.073.

Table 13. eTour with sim_3 result

Embedding	Max_size_sol	SN	MCN	P	R
Word2Vec	400	50	50	0.0589	0.065
		50	**100**	**0.0785**	**0.0747**
		60	60	0.0625	0.0617
GloVe	400	**60**	**60**	**0.0767**	**0.0812**
		50	50	0.063	0.054
		50	100	0.059	0.06
FastText	400	60	60	0.0438	0.0455
		50	**100**	**0.073**	**0.071**
		50	50	0.051	0.068

In Table *14* we summarize the best results obtained applying the three pretrained WE models on the three datasets using the three objective functions. Along with the P and R sores, we add the F1 measure in this summary table. F1 score is calculated following Equation (40); it is obtained by doubling the product of P and R scores divided by their sum.

$$F1 = 2\frac{P * R}{P + R} \tag{40}$$

The EBT dataset obtains the highest F1 score of 0.203 with the sim_1 function with the GloVe model. A comparable score of 0.195 is recorded from the sim_3 function with the GloVe model. The sim_1 function with the Word2Vec model obtains 0.186, and with the FastText model it scores 0.119. The sim_2 function with both the FastText and Word2Vec score comparable values of 0.108 and 0.113, respectively, whereas with the GloVe model it scores 0.126. The sim_3 with the Word2Vec scores 0.141, the nearest competitive score of 0.103 is obtained with the FastText model.

Table 14. Summary of experiment on three datasets

Datasets	Pretrained Model	sim_1			sim_2			sim_3		
		P	R	F1	P	R	F1	P	R	F1
EBT	Word2Vec	0.188	0.184	0.186	0.104	0.112	0.108	0.140	0.143	0.141
	GloVe	0.174	0.245	0.203	0.145	0.112	0.126	0.187	0.204	0.195
	FastText	0.102	0.143	0.119	0.105	0.122	0.113	0.104	0.102	0.103
EasyClinic	Word2Vec	0.102	0.097	0.099	0.095	0.097	0.096	0.101	0.097	0.099
	GloVe	0.088	0.106	0.096	0.090	0.086	0.088	0.076	0.075	0.076
	FastText	0.034	0.043	0.038	0.060	0.065	0.062	0.036	0.032	0.034
eTour	Word2Vec	0.096	0.091	0.093	0.087	0.071	0.078	0.079	0.075	0.077
	GloVe	0.114	0.123	0.118	0.073	0.075	0.074	0.077	0.081	0.079
	FastText	0.091	0.097	0.094	0.084	0.064	0.073	0.073	0.071	0.072

EasyClinic dataset records the same F1 score of 0.99 from the sim_1 and sim_3 with the Word2Vec model. The sim_2 with Word2Vec scored 0.096. The sim_1 with the GloVe model records the same F1 score of 0.96 and with the FastText it obtains 0.038. The sim_2 and sim_3 with the GloVe model are 0.088 and 0.076, respectively. While with the FastText model sim_2 scores 0.062, and sim_3 scores 0.034.

The eTour dataset obtains the highest F1 score of 0.118 with the sim_1 using the GloVe pretrained model. eTour obtains a comparable F1 score with sim_1 function using Word2Vec and FastText model; with Word2Vec the score is 0.093 and with FastText it is 0.094. For the other two objective functions the F1 scores obtained using all three pretrained models have a similar result. The sim_2 function with Word2Vec records 0.078, with GloVe the score is 0.074, and with FastText it is 0073. The objective function sim_3 scores 0.077, 0.079, and 0.072 with Word2Vec, GloVe, and FastText, respectively.

8. RESULTS AND DISCUSSION

Based on our experiment, we revisit the three research questions presented in Section 1.

RQ1: How does performance vary with each objective function?

The Table *15* and Figure *3* show the best F1 scores obtained by each dataset using the three similarity functions that we use with the ABC algorithm in this investigation.

Table 15. F1 scores of three datasets

Datasets	Pretrained Model	F1		
		sim_1	sim_2	sim_3
EBT	Word2Vec	0.186	0.108	0.141
	GloVe	0.203	0.126	0.195
	FastText	0.119	0.113	0.103
EasyClinic	Word2Vec	0.099	0.096	0.099
	GloVe	0.096	0.088	0.076
	FastText	0.038	0.062	0.034
eTour	Word2Vec	0.093	0.078	0.077
	GloVe	0.118	0.074	0.079
	FastText	0.094	0.073	0.072

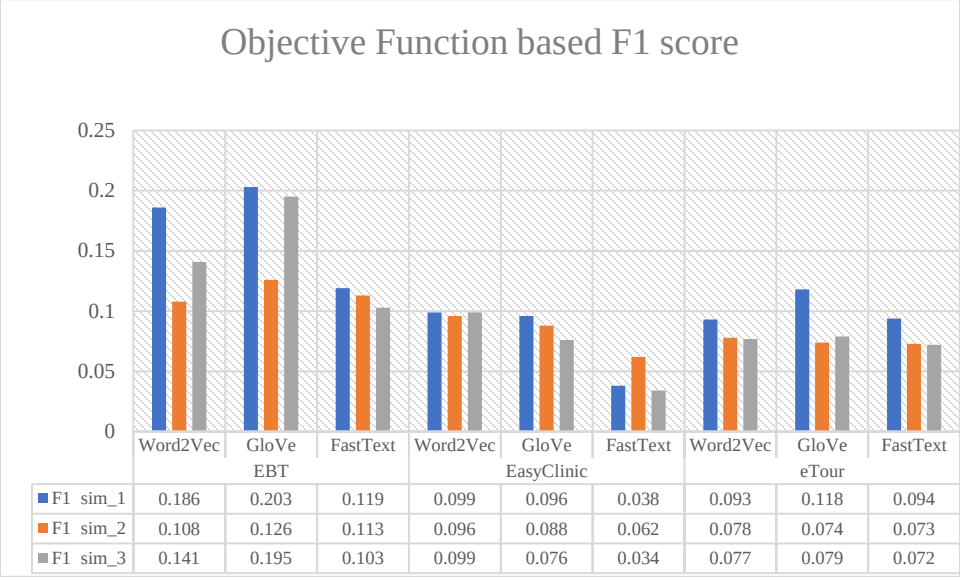

Figure 3. F1 scores of three datasets based on objective functions

The Figure *3* indicates that the objective function sim_1 outperforms the other two functions. The EBT dataset records the best F1 score of 0.203 with sim_1 function trained on GloVe model. The eTour dataset also records its highest F1 score of 0.118 with the same GloVe model-based sim_1 function. The EasyClinic dataset performs the same score of 0.99 with both the sim_1 and sim_3 functions.

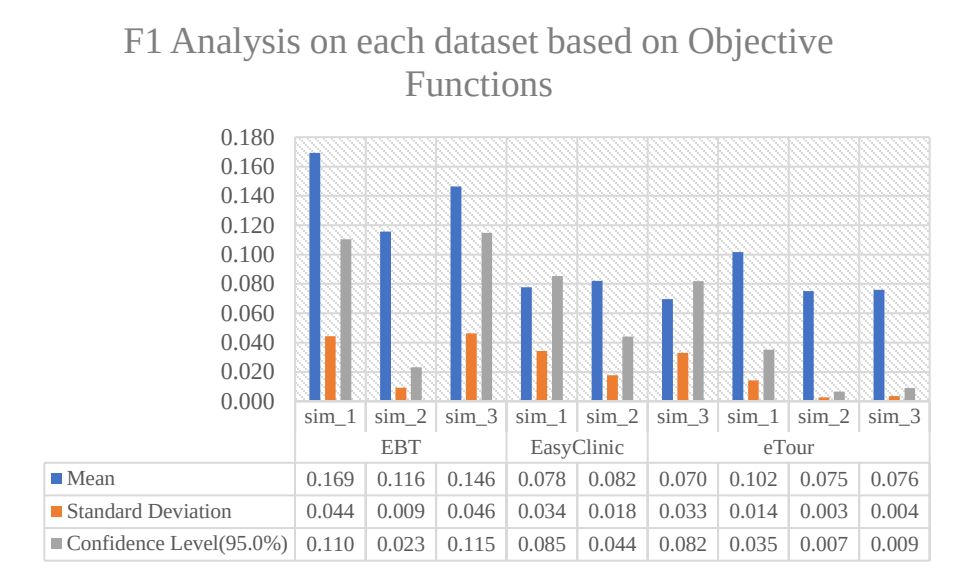

Figure 4. Comparison of three objective functions on each dataset

A comparison between the three objective functions based on their statistical significance is presented in Figure *4*. The Figure *4* represents the mean, standard deviation, and 95% confidence level error score of F1 obtained from each dataset using the three objective functions. The mean in Figure *4* is the sum of the three WE model's F1 scores from Table *15* under each similarity function divided by three (the number three represents the three WE models). For example, the highest mean F1 score of 0.169 is obtained by the EBT dataset with sim_1 function, and this score is measured by calculating the sum of F1 sim_1 scores (0.186+0.203+0.119), from Table *15* or Figure *3*, and dividing the sum by three. The standard deviation represents the dispersions of the F1 score, meaning how far the data spreads from the mean. The EBT sim_1 mean 0.169 has a standard deviation of 0.044, which indicates that the sim_3 mean score of

0.146 exists within one standard deviation of sim_1, but the sim_2 mean of 0.116 does not exist within one standard deviation of sim_1 mean. The error score of 95% confidence level provides the lower and upper limit of the F1 score. The EBT sim_1 mean has an error score of 0.11, which indicates that the lowest F1 score could be 0.059 and the highest could be 0.279.

The EasyClinic dataset achieved its highest mean with sim_2; however, the differences between the three means are not very significant as they all exist within one standard deviation of each other. Additionally, the standard deviation of sim_1 mean is higher than the other two means, which means that the values obtained from sim_1 are more spread out than sim_2. The 95% confidence level suggests that the best F1 score of 0.163 that could be obtained by the EasyClinic dataset is with the sim_1 function and the lowest score of 0.012 could be obtained with the sim_3 function.

The eTour dataset also scored the best mean of 0.102 with the sim_1 function. The lower means with low standard deviations of the other two functions indicates that they are not competitive enough with the sim_1. With a 95% confidence level, the highest F1 score that could be achieved by the eTour dataset is 0.137 and the lowest is 0.067.

A separate analysis on each dataset in Figure *4* confirms that the sim_1 function is the best performer among the three WE based objective functions.

RQ2. How does performance vary with different pretrained models?

We compare the three pretrained models based on the best F1 score obtained from each dataset. The Table *15* and Figure *5* present the F1 score of each dataset based on the three

pretrained WE models. The Figure *5* shows that the GloVe model records the highest F1 score for both EBT and eTour datasets. For EasyClinic, The Word2Vec model produces the highest score of 0.99, and with the score of 0.96, the GloVe model is the second highest scorer.

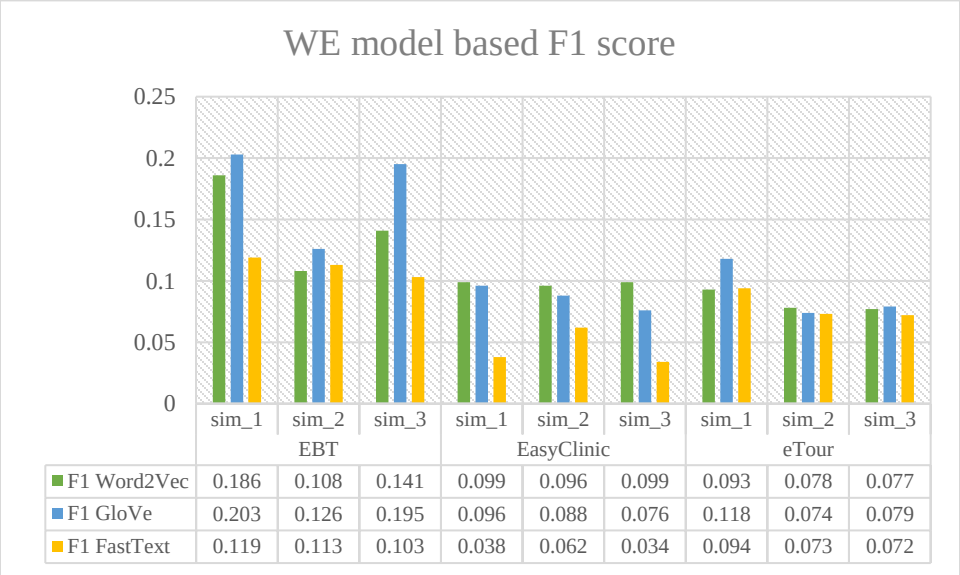

Figure 5. F1 scores of three datasets based on WE models

In Figure *6* we represent a statistical analysis of F1 score on each dataset based on the three pretrained models. The mean in Figure *6* is calculated by taking the sum of the F1 scores of the three objective functions on each WE model then divides that sum by three (the number three represents the number of objective functions).

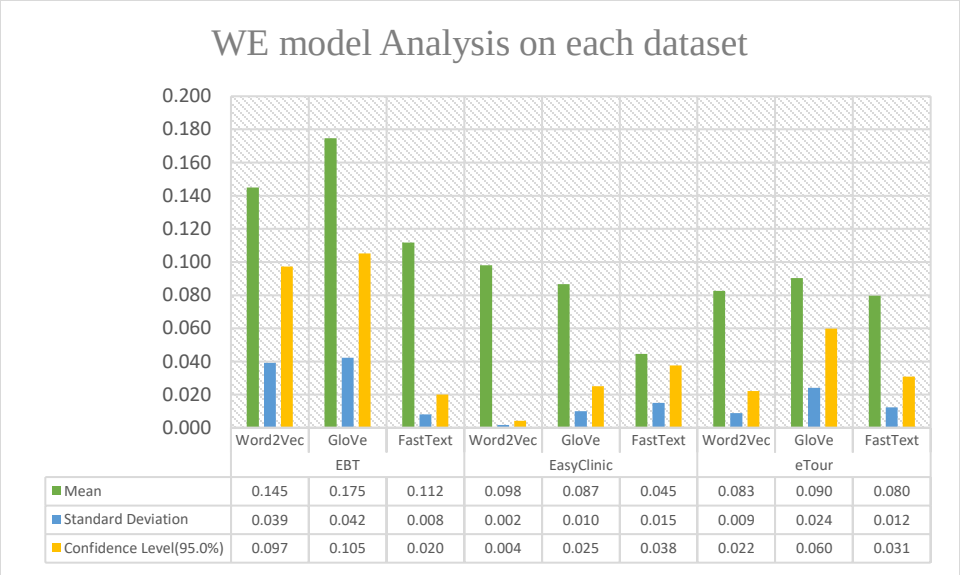

Figure 6. Comparison of three pretrained WE models on each dataset

The EBT dataset records the highest mean F1 score of 0.175 with the GloVe model, which has a standard deviation of 0.042, and with a 95% confidence level the error recorded is 0.105. The confidence interval is from 0.07 to 0.28. The closest competitor is the Word2Vec model with a score of 0.145 with a standard deviation of 0.039 and a confidence interval of (0.048, 0.242). The FastText model scored the lowest of 0.112 with a standard deviation of 0.008 and error score of 0.02, which gives a lower limit of 0.092 and upper limit of 0.132 for F1. This analysis indicates that the Word2Vec F1 score exists within one standard deviation of GloVe and vice versa. However, the FastText score does not exist within one standard deviation of the GloVe model. The FastText mean F1 score exists within one standard deviation of Word2Vec model score; however, the FastText F1's standard deviation of 0.008 indicates that the

Word2Vec F1 score is not within the three standard deviation range of FastText model. Therefor the EBT dataset records the GloVe model as the highest, the Word2Vec model as comparable with the GloVe model, and the FastText model as the lowest performer.

The EasyClinic dataset scores the highest mean F1 of 0.098 with the Word2vec model, which has a standard deviation score of 0.002 and an error of 0.004 with 95% confidence level. This finding indicates that the lower limit of mean F1 is 0.094 and the upper limit is 0.102. The next highest mean F1 score is obtained by the GloVe model, which is 0.087 with a standard deviation of 0.01 and an error value of 0.025. The mean score obtained by the GloVe model does not belong within one standard deviation of the Word2Vec mean and vice versa; however, the 95% confidence interval (0.062, 0.112) of the GloVe model indicates that the GloVe could achieve a higher F1 score of 0.112, which is greater than the upper limit of Word2Vec model, 0.102. The FastText scored the lowest mean of 0.045 with a standard deviation of 0.015 and an error score of 0.038. Therefore, for the EasyClinic dataset, both the GloVe and Word2Vec have a comparable performance.

The highest mean F1 of the eTour dataset is recorded with GloVe model and the score is 0.09 with a standard deviation of 0.024 and an error of 0.06, which gives the lower limit of 0.03 and the higher limit of 0.15. Both the Word2Vec and FastText model scored a comparable F1 mean of 0.083 and 0.08, respectively. The differences between the three means have low significance. The Word2Vec and FastText exist within one standard deviation of the GloVe model and vice versa. However, the 95% confidence level indicates that the highest possible

score of 0.15 could be achieved with the GloVe model, which suggests that the GloVe model is the best performer.

RQ3. How does a WE based objective function and an ABC combination perform in TLR task automation?

Based on our discussion on the previous two research questions, we can summarize that the WE based objective function, specifically the GloVe model based sim_1 objective function is the best performer. The model produces an F1 score of below 20%, which indicates that our three WE based objective functions and the ABC combination are not efficient for the purpose of TLR task automation.

The experimental result obtained from the EBT dataset is summarized in Table *16*. The P, R, and F1 scores obtained using the three similarity functions with three pretrained models are depicted in Figure *7*. The Figure *7* shows that the R score achieved is better than the P score. However, the R score of the EBT dataset resides below 25%, while the P and F1 scores are below 20%.

Table 16. P, R, and F1 scores of EBT dataset

Objective function	Pretrained model	P	R	F1
sim_1	Word2Vec	0.124	0.225	0.160
		0.113	0.174	0.137
		0.128	0.184	0.151
		0.113	0.143	0.126
		0.188	0.184	0.186
		0.167	0.184	0.175
		0.149	0.133	0.141
		0.171	0.122	0.143
	GloVe	0.119	0.214	0.153
		0.152	0.194	0.170
		0.169	0.245	0.200
		0.174	0.245	0.203
		0.135	0.133	0.134
	FastText	0.073	0.133	0.094
		0.102	0.143	0.119
		0.103	0.092	0.097
sim_2	Word2Vec	0.058	0.102	0.074
		0.045	0.061	0.052
		0.104	0.112	0.108
	GloVe	0.056	0.102	0.072
		0.145	0.112	0.126
		0.135	0.143	0.139
	FastText	0.080	0.123	0.097
		0.053	0.082	0.064
		0.105	0.122	0.113
sim_3	Word2Vec	0.090	0.153	0.113
		0.112	0.153	0.129
		0.140	0.143	0.141
	GloVe	0.136	0.245	0.175
		0.154	0.235	0.186
		0.187	0.204	0.195
	FastText	0.061	0.112	0.079
		0.066	0.092	0.077
		0.104	0.102	0.103

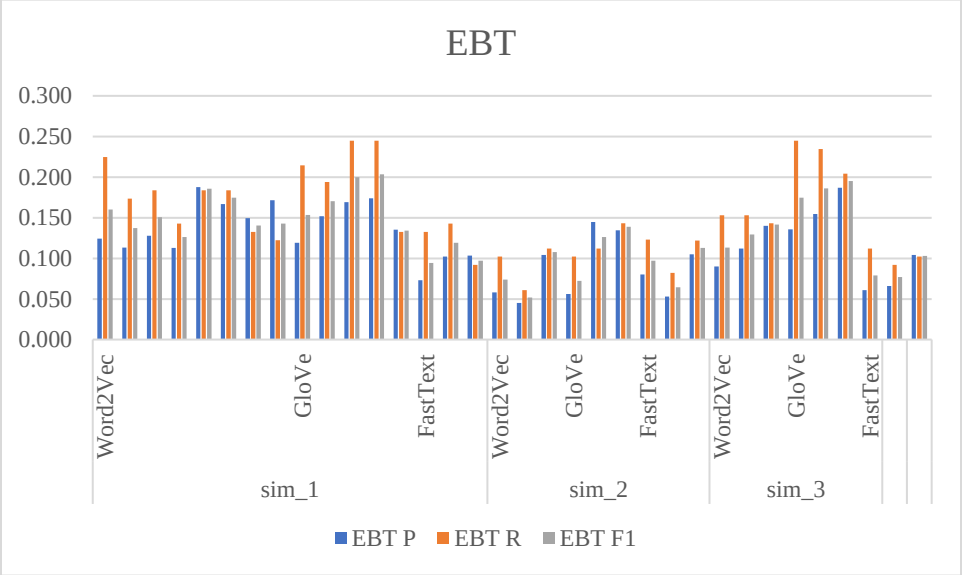

Figure 7. P, R, and F1 scores of EBT dataset

A summary of the result obtained by using the WE based model with the EasyClinic dataset is presented in Table *17*. The P, R, and F1 scores of EasyClinic dataset are depicted in Figure *8*. This dataset records a comparable P and R score. The highest R score is in 15% range and the P and F1 scores are within 10%.

Table 17. P, R, and F1 scores of EasyClinic dataset

Objective function	Pretrained model	P	R	F1
sim_1	Word2Vec	0.057	0.065	0.060
		0.102	0.097	0.099
		0.086	0.108	0.096
		0.085	0.086	0.085
		0.063	0.065	0.064
		0.073	0.054	0.062
	GloVe	0.071	0.149	0.096
		0.065	0.054	0.059
		0.058	0.054	0.056
		0.088	0.106	0.096
	FastText	0.021	0.043	0.028
		0.042	0.032	0.036
		0.034	0.043	0.038
sim_2	Word2Vec	0.095	0.097	0.096
		0.065	0.065	0.065
		0.071	0.085	0.078
	GloVe	0.090	0.086	0.088
		0.075	0.052	0.062
		0.074	0.054	0.062
	FastText	0.060	0.065	0.062
		0.056	0.055	0.055
		0.060	0.044	0.051
sim_3	Word2Vec	0.101	0.097	0.099
		0.078	0.054	0.064
		0.057	0.043	0.049
	GloVe	0.076	0.075	0.076
		0.056	0.043	0.049
		0.044	0.032	0.037
	FastText	0.036	0.032	0.034
		0.015	0.011	0.013
		0.030	0.022	0.025
		0.021	0.011	0.014

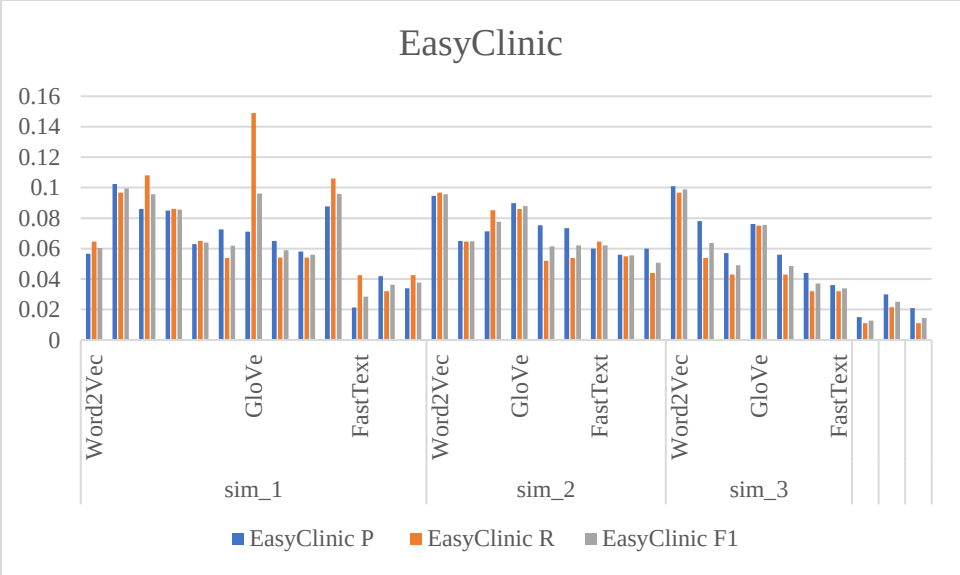

Figure 8. P, R, and F1 scores of EasyClinic dataset

The eTour dataset result is summarized in Table *18* and illustrated in Figure *9*. The P and R score obtained using the WE based model on this dataset are comparable. The highest R is in the range of 12% and the P and F1 scores are within 11%.

Table 18. P, R, and F1 scores of eTour dataset

Objective function	Pretrained model	P	R	F1
sim_1	Word2Vec	0.070	0.078	0.074
		0.096	0.091	0.093
		0.061	0.062	0.061
		0.053	0.042	0.047
		0.088	0.068	0.077
	GloVe	0.114	0.123	0.118
		0.066	0.068	0.067
		0.064	0.058	0.061
		0.088	0.068	0.077
	FastText	0.091	0.097	0.094
		0.050	0.052	0.051
		0.031	0.029	0.030
		0.067	0.055	0.060
sim_2	Wor2Vec	0.061	0.065	0.063
		0.052	0.052	0.052
		0.073	0.071	0.072
		0.087	0.071	0.078
	GloVe	0.067	0.075	0.071
		0.073	0.075	0.074
		0.073	0.071	0.072
		0.062	0.042	0.050
	FastText	0.084	0.064	0.073
		0.060	0.062	0.061
		0.049	0.042	0.045
		0.046	0.036	0.040
sim_3	Word2Vec	0.059	0.065	0.062
		0.079	0.075	0.077
		0.063	0.062	0.062
	GloVe	0.077	0.081	0.079
		0.063	0.054	0.058
		0.059	0.060	0.059
	FastText	0.044	0.046	0.045
		0.073	0.071	0.072
		0.051	0.068	0.058

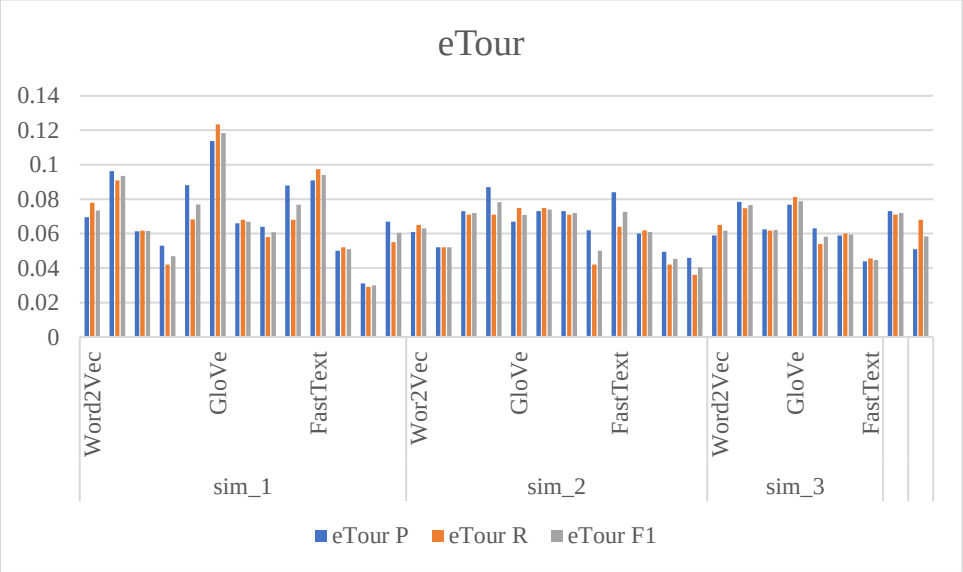

Figure 9. P, R, and F1 scores of eTour dataset

Based on the results of the three datasets, we conclude that neither of the three WE based objective functions and the ABC algorithm applied to the three datasets performs at an acceptable level for TLR task automation.

9. CONCLUSIONS AND FUTURE WORK

We investigate the compatibility of a WE based ABC algorithm applied to TLR. We use three different WE based objective functions and open source pretrained models: Word2Vec, GloVe, and FastText to learn the document vectors. We evaluate the performance of these pretrained WE models combined with the ABC algorithm for automating TLR. Our analysis on the three datasets suggests that the GloVe model performs slightly better than the Word2Vec model, and the FastText has the lowest performance rate. The objective function sim_1 performs slightly better than the other two objective functions. The R was better than the P; however, the best F1 mean achieved using the three datasets was below 0.2, which indicates that the WE based ABC does not perform at an acceptable level for TLR task automation.

For future experimentation, there is potential for improvement given that the ABC algorithm involves extensive parameter adjustment which demands more experimentation. Additionally, changing the preprocessing of the datasets might result in improvement. Other future exploration involves building a domain specific customized word embedding to advance the research. Three other embedding systems, recently gaining popularity in NLP are ELMO (Peters et al., 2018), BERT (Kenton, Kristina, & Devlin, 2019), and VAMPIRE (Gururangan, Dang, Card, & Smith, 2019). We want to explore their applicability and performance in the TLR task automation process. In addition, we want to investigate with other possible objective functions.

REFERENCES

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., & Merlo, E. (2002). Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, *28*(10), 970–983. https://doi.org/10.1109/TSE.2002.1041053

Bella, E. E., Creff, S., Gervais, M. P., & Bendraou, R. (2019). ATLaS: A framework for traceability links recovery combining information retrieval and semi-supervised techniques. *Proceedings - 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference, EDOC 2019*, 161–170. https://doi.org/10.1109/EDOC.2019.00028

Bengio, Y., Ducharme, R., & Vincent, P. (2001). A neural probabilistic language model. *Advances in Neural Information Processing Systems*, *3*, 1137–1155.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146. https://doi.org/10.1162/tacl_a_00051

Cheng, S., Yan, X., & Khan, A. A. (2020). A Similarity Integration Method based Information Retrieval and Word Embedding in Bug Localization. *Proceedings - 2020 IEEE 20th International Conference on Software Quality, Reliability, and Security, QRS 2020*, (October), 180–187. https://doi.org/10.1109/QRS51102.2020.00034

Coest.org. (n.d.). No Title. Retrieved April 16, 2021, from http://coest.org/

Goldberg, Y., & Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method, (2), 1–5. Retrieved from http://arxiv.org/abs/1402.3722

Guo, J., Cheng, J., & Cleland-Huang, J. (2017). Semantically Enhanced Software Traceability Using Deep Learning Techniques. *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017*, 3–14. https://doi.org/10.1109/ICSE.2017.9

Gururangan, S., Dang, T., Card, D., & Smith, N. A. (2019). Variational Pretraining for Semi-supervised Text Classification.

Holbrook, E. A., Hayes, J. H., Dekhtyar, A., & Li, W. (2013). A study of methods for textual satisfaction assessment. *Empirical Software Engineering*, *18*(1), 139–176. https://doi.org/10.1007/s10664-012-9198-8

Karaboga, D., & Basturk, B. (2007). Artificial Bee Colony (ABC) Optimization Algorithm for

Solving Constrained Optimization. *IEEE*, 789–798.

Kenton, M. C., Kristina, L., & Devlin, J. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, (Mlm). Retrieved from https://arxiv.org/pdf/1810.04805v2.pdf

Lam, A. N., Nguyen, A. T., Nguyen, H. A., & Nguyen, T. N. (2017). Bug Localization with Combination of Deep Learning and Information Retrieval. *IEEE International Conference on Program Comprehension*, 218–229. https://doi.org/10.1109/ICPC.2017.24

Liu, N., Chan, A., Feng, C., Fulton, J., & Wu, S. (2019). Automate RFP response generation process using fasttext word embeddings and soft cosine measure. *ACM International Conference Proceeding Series*, 12–17. https://doi.org/10.1145/3349341.3349362

Mihalcea, R., Corley, C., & Strapparava, C. (2006). Corpus-based and knowledge-based measures of text semantic similarity. *Proceedings of the National Conference on Artificial Intelligence*, *1*, 775–780.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Distributed Representations of Words and Phrases and their Compositionality arXiv : 1310 . 4546v1 [cs . CL] 16 Oct 2013. *ArXiv Preprint ArXiv:1310.4546*, *26*, 1–9.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013b). Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 1–12.

Nguyen, T. Van, Nguyen, A. T., Phan, H. D., Nguyen, T. D., & Nguyen, T. N. (2017). Combining Word2Vec with revised vector space model for better code retrieval. *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, 183–185. https://doi.org/10.1109/ICSE-C.2017.90

Niu, N., & Mahmoud, A. (2012). Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited. *2012 20th IEEE International Requirements Engineering Conference, RE 2012 - Proceedings*, 81–90. https://doi.org/10.1109/RE.2012.6345842

Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshynanyk, D., & De Lucia, A. (2013). How to effectively use topic models for software engineering tasks? An approach based on Genetic Algorithms. *Proceedings - International Conference on Software Engineering*, 522–531. https://doi.org/10.1109/ICSE.2013.6606598

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 1532–1543.

https://doi.org/10.3115/v1/d14-1162

Peter F. Brown, DeSouza, P. V., Mercer, R. L., Pietra, V. J. Della, & Lai, J. C. (1992). Class-Based n-gram Models of Natural Language. *Computational Linguistics*, *18*(4), 467–479. Retrieved from http://dl.acm.org/citation.cfm?id=176316

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, *1*, 2227–2237. https://doi.org/10.18653/v1/n18-1202

Rodriguez, D. V, & Carver, D. L. (2020). An IR-Based Artificial Bee Colony Approach for Traceability Link Recovery. *IEEE International Conference on Software Maintenance, ICSM*, *32*, 1145–1153. https://doi.org/10.1109/ICTAI50040.2020.00174

Rong, X. (2014). word2vec Parameter Learning Explained. Retrieved from http://arxiv.org/abs/1411.2738

Scott, D., T, D. S., W, F. G., K, L. T., & Richard, H. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*(6), 391–407. Retrieved from papers2://publication/uuid/BA23C102-A2EA-4493-AEA4-2B85C3BD3FA1

Sommerville, I. (2016). *Software engineering (10th edition)*. *Pearson Education Limited*.

Tian, Q., Cao, Q., & Sun, Q. (2019). Adapting Word Embeddings to Traceability Recovery. *Proceedings of 2018 International Conference on Information Systems and Computer Aided Education, ICISCAE 2018*, 255–261. https://doi.org/10.1109/ICISCAE.2018.8666883

Wang, Xiaobo, Lai, G., & Liu, C. (2009). Recovering Relationships between Documentation and Source Code based on the Characteristics of Software Engineering. *Electronic Notes in Theoretical Computer Science*, *243*, 121–137. https://doi.org/10.1016/j.entcs.2009.07.009

Wang, Xinye, Cao, Q., & Sun, Q. (2019). An Improved Approach based on Balanced Keyword Weight to Traceability Recovery. *IOP Conference Series: Materials Science and Engineering*, *569*(5). https://doi.org/10.1088/1757-899X/569/5/052109

Xiao, Y., Keung, J., Mi, Q., & Bennin, K. E. (2018). Improving Bug Localization with an Enhanced Convolutional Neural Network. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, *2017-Decem*, 338–347. https://doi.org/10.1109/APSEC.2017.40

Ye, X., Bunescu, R., & Liu, C. (2014). Learning to rank relevant files for bug reports using domain knowledge. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, *16-21-Nove*, 689–699. https://doi.org/10.1145/2635868.2635874

Ye, X., Shen, H., Ma, X., Bunescu, R., & Liu, C. (2016). From word embeddings to document similarities for improved information retrieval in software engineering. *Proceedings - International Conference on Software Engineering*, *14-22-May-*, 404–415. https://doi.org/10.1145/2884781.2884862

Zhao, T., Cao, Q., & Sun, Q. (2018). An Improved Approach to Traceability Recovery Based on Word Embeddings. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, *2017-Decem*, 81–89. https://doi.org/10.1109/APSEC.2017.14

Zheng, Y., Shi, Y., Guo, K., Li, W., & Zhu, L. (2017). Enhanced word embedding with multiple prototypes. *4th International Conference on Industrial Economics System and Industrial Security Engineering, IEIS 2017*, (91546201). https://doi.org/10.1109/IEIS.2017.8078651

Zhou, J., Zhang, H., & Lo, D. (2012). Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. *Proceedings - International Conference on Software Engineering*, 14–24. https://doi.org/10.1109/ICSE.2012.6227210

VITA

Mahfuza Khatun is from Bangladesh. She received her Bachelor of Science and Master of Science (Thesis) degree in Physics from the University of Dhaka, Bangladesh. She is now on her way to pursue a second Master of Science in Computer Science (Thesis) from Louisiana State University (LSU), Baton Rouge. Her area of concentration is Software Engineering. She has been working as a Graduate Assistant with the Education Support Service Team at Cox Communication Academic Center for Student-Athlete (CCACSA), LSU. She acknowledges the CCACSA, LSU family's support towards her endeavor during her entire period as a graduate student at LSU. The knowledge and experiences gathered from this institution have elevated both of her personal and profession lives. She is now focusing to serve a greater community.